Multi-Level Security in Multiagent Systems

Gerd Wagner gw@inf.fu-berlin.de

Inst.f.Informatik, Univ. Leipzig, Augustusplatz 10-11, 04109 Leipzig, Germany,
Tel+Fax: (+49 30) 834 95 69.

Abstract. Whenever agents deal with confidential information, it is important that they comply with a principled security policy. We show how the database concept of *multi-level security* can be applied to inter-agent communication. This includes the case where an unauthorized agent is misinformed on purpose in order to protect confidential information.

1 Introduction

In certain applications, it is essential to protect confidential information from unauthorized access. While access restrictions in relational databases have to be defined within the database schema, implying that only entire tables (i.e. predicates) can be protected, the concept of *multi-level security (MLS)*[1] allows to protect single rows of a table (i.e. atomic sentences) according to their security classification. The MLS concept is not concerned with lower-level security issues such as authentification, or secure message transport protocols, but only with the definition of secure query answering and secure update in information systems.

We show how multi-level security can be achieved in multi-agent and multi-database systems. This requires that the security restrictions defined in MLS tables have to be taken into consideration in inter-agent communication. We formalize multi-level security on the basis of our theory of vivid knowledge and agent systems introduced in [Wag95, Wag96].

A *vivid agent* is a software-controlled system whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. The basic functionality of a vivid agent comprises a knowledge system (including an update and an inference operation), and the capability to represent and perform actions in order to be able to generate and execute plans. Since a vivid agent is 'situated' in an environment with which it has to be able to communicate, it also needs the ability to react in response to perception events, and in response to communication events created by the communication acts of other agents. Reactions may be immediate and independent from the current believe state of the agent but they may also depend on the result of deliberation. In any case, they are triggered by events which are not controlled by the agent.

We do not assume a fixed formal language and a fixed logical system for the knowledge base of an agent. Rather, we believe that it is more appropriate to choose a suitable knowledge system for each agent individually according to its

[1] See, e.g., [Lan81, JS91, SWQ94].

domain and its tasks. In simple cases, a relational database-like system (admitting of atomic sentences only) will do the job, while in more involved cases one may need the ability to process, in addition to simple facts, uncertain, temporal or confidential information, or even such advanced capabilities as deductive query answering and abductive reasoning.

While certain agents may have rather limited capabilities, others are quite complex. We call the simplest form of a vivid agent a *reagent*. A reagent does not have explicit goals and intentions but only beliefs about the current state of affairs. It reacts to events in its environment, taking into account what it currently believes. A reagent updates its beliefs and draws inferences from them for answering queries by applying the respective operations of the vivid knowledge system it is based on.

A cooperative knowledge base can be viewed as a reagent, since cooperative query answering can be achieved on the basis of reactive communication protocols defined at design time (without planning for user-defined tasks communicated at run time). A multidatabase (MDB) system, involving inhomogeneous nodes with a global distribution schema, can therefore be conceptualized as a multi-reagent system. Notice that if there were only relational databases in a MDB, there would be no communication, since the semantics of RDBs assumes their completeness, i.e. a standard RDB has never any reason to ask another database for additional information. Communication between databases requires incomplete knowledge.

2 Vivid Knowledge Systems

The knowledge system of a vivid agent is based on three specific languages: L_{KB} is the set of all admissible knowledge bases, L_{Query} is the query language, and L_{Input} is the set of all admissible inputs, i.e. those formulas representing new information a KB may be updated with. While the input language defines what the agent can be told (i.e. what it is able to assimilate into its KB), the query language defines what the agent can be asked. Where L is a set of formulas, L^0 denotes its restriction to closed formulas (sentences). Elements of L^0_{Query}, i.e. closed query formulas, are also called *if-queries*.

Definition 1 (Knowledge System) *An abstract knowledge system*[2] $\boldsymbol{K}$ *consists of three languages and two operations: a knowledge representation language* L_{KB}*, a query language* L_{Query}*, an input language* L_{Input}*, an inference relation* $\vdash$*, such that* $X \vdash F$ *holds if* $F \in L^0_{\mathrm{Query}}$ *can be inferred from* $X \in L_{\mathrm{KB}}$*, and an update operation* Upd*, such that the result of updating* $X \in L_{\mathrm{KB}}$ *with* $F \in L^0_{\mathrm{Input}}$ *is the knowledge base* $\mathsf{Upd}(X, F)$*.*

[2] See also [Wag95].

Definition 2 (Answer Operation) *The answer operation* Ans *is defined for if-queries F by*

$$\mathsf{Ans}(X,F) = \begin{cases} \mathsf{yes} & \textit{if} \quad X \vdash F \\ \mathsf{no} & \textit{if} \quad X \vdash \neg F \\ \mathsf{unknown} & \textit{otherwise} \end{cases}$$

and for query formulas G with free variables $x_1, \ldots, x_n$ by

$$\langle c_1, \ldots, c_n \rangle \in \mathsf{Ans}(X, G(x_1, \ldots, x_n)) \quad \textit{if} \quad X \vdash G(c_1, \ldots, c_n)$$

We now present two important examples of knowledge systems: relational databases, and MLS databases.

2.1 Relational Databases

A relational database is a finite set of finite relations (tables) corresponding to a finite set of atomic sentences. For instance, a hospital database may contain facts expressing who is currently a patient and what are their diagnoses, $\Delta_{hosp} = \{\textit{Patient}, \textit{Diagnosis}\}$, where

$$\text{Patient} = \boxed{\begin{array}{l}\text{BY}\\ \text{MJ}\\ \text{JB}\end{array}} \quad \text{Diagnosis} = \boxed{\begin{array}{ll}\text{BY} & \text{alc}\\ \text{JB} & \text{mal}\end{array}}$$

The propositional representation of Δ_{hosp} is

$$X_{hosp} = \{p(BY), p(MJ), p(JB), d(BY, alc), d(JB, mal)\}$$

representing the information that Boris Yeltsin (BY), Michael Jackson (MJ), and James Bond (JB) are currently patients of the hospital, and the diagnosis of BY is alcoholism, and that of JB is malaria. As a kind of natural deduction from positive facts an inference relation $\vdash$ between a relational database X and an if-query is defined in the following way:

$$\begin{array}{ll}(a) & X \vdash a \text{ if } a \in X \\ (\neg a) & X \vdash \neg a \text{ if } a \notin X\end{array}$$

Notice the non-monotonicity of $(\neg a)$: negation in relational databases corresponds to *negation-as-failure*. Compound if-queries, involving conjunction and disjunction, are handled in the standard way. Negated compound if-queries are treated by simplification according to the DeMorgan rules and double negation elimination. We obtain, for example, $X_{hosp} \vdash p(MJ) \wedge \neg d(MJ, alc)$. Because of its built-in general Closed-World Assumption, a relational database X answers an if-query F by either yes or no: the answer is yes if $X \vdash F$, and no otherwise. For instance, $\mathsf{Ans}(X_{hosp}, d(BY, mal)) = \mathsf{no}$. Updates are insertions, $\mathsf{Upd}(X, a) := X \cup \{a\}$, and deletions, $\mathsf{Upd}(X, \neg a) := X - \{a\}$, where a is an atom. For a consistent set of literals E, we have $\mathsf{Upd}(X, E) = X \cup E^+ - E^-$,

where E^+ contains the positive, and E^- contains the negative literals of E. For instance,

$$\begin{aligned}&\mathsf{Upd}(X_{hosp}, \neg d(JB, mal) \wedge d(JB, yf))\\ &= \{p(BY), p(MJ), p(JB), d(BY, alc), d(JB, yf)\}\end{aligned}$$

describes a possible transaction. The knowledge system of relational databases is denoted by $\boldsymbol{A}$ (for $\boldsymbol{A}$tomic). We also describe a knowledge system by means of its language table:

$\boldsymbol{A}$ =

L_{KB}	2^{At}
L_{Query}	$L(\neg, \wedge, \vee, \exists, \forall)$
L^0_{Ans}	{yes, no}
L_{Input}	Lit

Knowledge systems extending $\boldsymbol{A}$ conservatively are called *vivid*. Positive vivid knowledge systems use a general Closed-World Assumption, whereas general vivid knowledge systems employ specific Closed-World Assumptions (and possibly two kinds of negation). For instance, $\boldsymbol{A}$ can be extended to the general vivid knowledge system of *factbases*, by allowing for literals instead of atoms as information units. Further important examples of positive vivid knowledge systems are temporal, uncertain and MLS databases. All these kinds of knowledge bases can be extended to *deductive knowledge bases* by adding deduction rules of the form $F \leftarrow G$ [Wag95].

2.2 MLS Databases

In *multi-level secure (MLS)* databases,[3] all information items are assigned a security classification, and all database users are assigned a security clearance, both from a partially ordered set of security levels. For instance, the four security levels *unclassified* (0), *confidential* (1), *secret* (2), and *top secret* (3) may be used to classify entries in a MLS table.

As an example, consider the database of a hospital. Depending on the respective person it may be sensitive information to know whether someone is a patient in the hospital. In the case of a politician, such infomation would be publicly available. But not so in the case of a shy pop star, or a secret service agent. The following MLS tables containing the records of patients and their diagnoses form the hospital database X_{hosp}:

Patient =

BY	0
MJ	1
JB	3

Diagnosis =

BY alc	1
JB mal	3

These tables represent the following beliefs at the respective clearance level:

level	beliefs
0	$\{p(BY)\}$
1, 2	$\{p(BY), p(MJ), d(BY, alc)\}$
3	$\{p(BY), p(MJ), p(JB), d(BY, alc), d(JB, mal)\}$

[3] See, e.g., [Lan81, JS91, SWQ94].

The basic principle underlying MLS query answering is called *simple security property* and was defined in the Bell-LaPadula model of *mandatory security*, see [Lan81]. It can also be described by the slogan "no read up", i.e. users are only permitted to read from a level dominated by their own. We will formalize this principle below in the definition of secure inference. As opposed to [SWQ94], we think that it should be defined by the logical semantics of MLS databases how lower-level beliefs carry over to higher-level beliefs.

Notice that it is not possible to preserve privacy and maintain security by simply omitting information, like in the reply *'no answer'* to the question *'Is Michael Jackson a patient in this hospital ?'*. The asking reporter could easily infer from this refusal to answer that MJ must be a patient in the hospital. The only way to maintain security is to give a wrong answer, i.e. to misinform the unauthorized asker. The rationality principle of secure inference is the

(Principle of Minimal Misinformation) Askers are only misinformed about an information item if they are not sufficiently authorized with respect to that item.

Assume, for instance, that Boris Yeltsin is in the hospital with an accute alcoholism. When a reporter asks if BY is in the hospital, he receives the answer *yes*. If he then asks whether BY has drunk too much, the secure answer may be *no* or *unknown*.[4] When being asked what BY suffers from, the hospital information system may reply to the reporter that he has a severe influenca (i.e. a *cover story*).

MLS queries are annotated by the clearance level of the asker. Since reporters have clerance level 0 (unclassified), we get

$$\mathsf{Ans}(X_{hosp},\, p(MJ)/0) = \mathsf{no}$$

while a doctor of the hospital with clerance level 2 would get the right answer: *'yes, MJ is a patient in this hospital'*,

$$\mathsf{Ans}(X_{hosp},\, p(MJ)/2) = \mathsf{yes}$$

Definition 3 (Security Hierarchy) *A* security hierarchy *SH is a finite partial order with a greatest element denoted by $\top$.*

In our examples, we will only use the security hierarchy $\{0, 1, 2, 3\}$ introduced above.

Definition 4 (MLS Table) *A* MLS table *R over a security hierarchy SH and a relation schema $r(x_1, \ldots, x_n)$ is a finite subset of $D_1 \times \ldots \times D_n \times SH$.*

[4] Notice that this choice in misinforming is only available in the case of incomplete predicates allowing for the answer *unknown*.

Definition 5 (MLS Database) *A MLS database Δ over a schema $\Sigma = \langle\{r_1, \ldots, r_m\}, SH\rangle$ is a finite set of MLS tables $\{R_1, \ldots, R_m\}$ over the security hierarchy SH. Its propositional representation is*

$$X_\Delta = \bigcup_{i=1}^{m} \{r_i(\mathbf{c})/\lambda : \langle \mathbf{c}, \lambda\rangle \in R_i\}$$

It can be decomposed into a set of relational databases $\{\Delta^\lambda \mid \lambda \in SH\}$, such that

$$\begin{array}{l} \Delta^\lambda = \{R_1^\lambda, \ldots, R_m^\lambda\} \\ R_i^\lambda = \{\mathbf{c} \mid \langle \mathbf{c}, \kappa\rangle \in R_i \ \ \& \ \ \kappa \leq \lambda\} \end{array}$$

We write X^λ, instead of X_{Δ^λ}, for the propositionl representation of Δ^λ.

It may be useful to be able to ask questions relative to others' viewpoints. For example, the nurse (with clearance level 1) might need to ask, *'If a reporter (with clearance level 0) asks for a list of the current patients, what will be the answer ?'* She would put this as the query:

$$\mathsf{Ans}(X_{hosp}, (\mathsf{B}_0 p(x))/1) \ = \ \{BY\}$$

There is no need to allow for nested B operators.

Definition 6 (Secure Inference) *Let X be a MLS database, $\lambda, \kappa \in SH$, $l \in$ Lit, $F, G \in L(\neg, \wedge, \vee, \exists, \forall, \mathsf{B}_\lambda)$, and $H \in L^0(\neg, \wedge, \vee, \exists, \forall)$.*

$$\begin{array}{lrcl} (l) & X \vdash l/\lambda & :\Longleftrightarrow & X^\lambda \vdash_A l \\ (\wedge) & X \vdash (F \wedge G)/\lambda & :\Longleftrightarrow & X \vdash F/\lambda \ \ \& \ \ X \vdash G/\lambda \\ (\vee) & X \vdash (F \vee G)/\lambda & :\Longleftrightarrow & X \vdash F/\lambda \ \ \textit{or} \ \ X \vdash G/\lambda \\ (\exists) & X \vdash (\exists x F(x))/\lambda & :\Longleftrightarrow & \mathsf{Ans}(X, F(x)/\lambda) \neq \emptyset \\ (\forall) & X \vdash (\forall x F(x))/\lambda & :\Longleftrightarrow & \mathsf{Ans}(X, \neg F(x)/\lambda) = \emptyset \\ (\mathsf{B}_\kappa) & X \vdash (\mathsf{B}_\kappa H)/\lambda & :\Longleftrightarrow & \kappa \leq \lambda \ \ \& \ \ X^\kappa \vdash_A H \end{array}$$

where $\vdash_A$ is inference in $\boldsymbol{A}$.

In formalizing secure update, we do not follow the Bell-LaPadula model which requires a "no write down" policy (also called '$*$-property') where users are only permitted to write to a level that dominates their own. This principle is supposed to prevent users from passing information directly downward through the security hierarchy. It makes only sense, however, in an intelligence context where a highly authorized user may be spy. In a security policy for normal organizations without particular intelligence concerns it seems more reasonable to assume that users in the higher level of the hierarchy can be trusted not to disclose sensitive information to lower levels. This also corresponds more closely to management practice. We will therefore assume the principle of "no write up" preventing users to overwrite information at levels above their own while permitting them to update lower level information either seriously or for the purpose of misinformation.

Inputs to MLS databases are annotated by the clearance level of the information supplier.

Definition 7 (Secure Update) *Let a be an atom, and l a literal.*

$$\begin{aligned}
\mathsf{Upd}(X, (\mathsf{B}_\kappa a)/\lambda) &:= \begin{cases} X \cup \{a/\kappa\} & \textit{if } \kappa \leq \lambda \ \ \& \ \ X \not\vdash a/\kappa \\ X & \textit{otherwise} \end{cases} \\
\mathsf{Upd}(X, (\mathsf{B}_\kappa \neg a)/\lambda) &:= X - \{a/\mu \in X : \mu \leq \kappa \leq \lambda\} \\
\mathsf{Upd}(X, l/\lambda) &:= \mathsf{Upd}(X, (\mathsf{B}_\lambda l)/\lambda)
\end{aligned}$$

The knowledge system of MLS databases, denoted by $\boldsymbol{SA}$, is then defined as

$$\boldsymbol{SA} = \begin{array}{|l|l|} \hline L_{\mathrm{KB}} & 2^{\mathrm{At} \times SH} \\ \hline L^0_{\mathrm{Query}} & L^0(\neg, \wedge, \vee, \exists, \forall, \mathsf{B}_\kappa) \times SH \\ \hline L^0_{\mathrm{Ans}} & \{\mathsf{yes}, \mathsf{no}\} \\ \hline L_{\mathrm{Input}} & \mathrm{Lit} \times SH \cup \mathsf{B}_\kappa \mathrm{Lit} \times SH \\ \hline \end{array}$$

Notice that in MLS databases, it is not possible to protect negative information by providing suitable misinformation. If, for instance, a hospital has to pretend that James Bond is among its patients, i.e. if the negative information $\neg p(JB)$ has to be protected, say at clearance level 3 (top secret), this cannot be achieved by means of simple MLS tables which would have to record negative entries in addition to positve ones. This is possible, however, in *MLS bitables* which are defined in [Wag97].

3 Specification and Execution of Reagents

Simple vivid agents whose mental state comprises only beliefs, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $\mathcal{A} = \langle X, EQ, RR \rangle$, on the basis of a knowledge system $\boldsymbol{K}$, consists of

1. a knowledge base $X \in L_{\mathrm{KB}}$,
2. an event queue EQ being a list of instantiated event expressions, and
3. a set RR of *reaction rules*, consisting of epistemic and physical reaction and interaction rules which code the reactive and communicative behavior of the agent.

A multi-reagent system is a tuple of reagents: $\mathcal{S} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$.

3.1 Operational Semantics of Reaction Rules

Reaction rules encode the behavior of vivid agents in response to perception events created by the agent's perception subsystems, and to communication events created by communication acts of other agents. We distinguish between epistemic, physical and communicative reaction rules. We use L_{PEvt} and L_{CEvt} to denote the perception and communication event languages, and $L_{\mathrm{Evt}} = L_{\mathrm{PEvt}} \cup L_{\mathrm{CEvt}}$. The following table describes the different formats of epistemic, physical and communicative reaction rules:

epistemic	$\mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$
physical	$\mathsf{do}(\alpha),\ \mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$
communicative	$\mathsf{sendMsg}[\eta, R],\ \mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$

The event condition $\mathsf{recvMsg}[\varepsilon(U), S]$ is a test whether the event queue of the agent contains a message of the form $\varepsilon(U)$ sent by some perception subsystem of the agent or by another agent identified by S, where $\varepsilon \in L_{\mathrm{Evt}}$ represents a perception or a communication event type, and U is a suitable list of parameters. The epistemic condition $\mathit{Cond} \in L_{\mathrm{Query}}$ refers to the current knowledge state, and the epistemic effect $\mathit{Eff} \in L_{\mathrm{Input}}$ specifies an update of the current knowledge state.

In a physical reaction, $\mathsf{do}(\alpha(V))$ calls a procedure realizing the action α with parameters V. In a communicative reaction, $\mathsf{sendMsg}[\eta(V), R]$ sends the message $\eta \in L_{\mathrm{CEvt}}$ with parameters V to the receiver R. Both perception and communication events are represented by incoming messages. We identify a communication act with the corresponding communication event which is perceived by the addressee of the communication act.

Reaction rules are triggered by events. The agent interpreter continuously checks the event queue of the agent. If there is a new event message, it is matched with the event condition of all reaction rules, and the epistemic conditions of those rules matching the event are evaluated. If they are satisfiable in the current knowledge base, all free variables in the rules are instantiated accordingly resulting in a set of triggered actions with associated epistemic effects. All these actions are then executed, leading to physical actions and to sending messages to other agents, and their epistemic effects are assimilated into the current knowledge base.

3.2 Defining the Execution of Reagents

The *perception-reaction-cycle* in the execution of reagents consists of the following steps:

repeat

1. Get the next message from the event queue, and check whether it triggers any reaction rules. If it does not, then repeat 1, else continue.
2. For each of the triggered reaction rules, assimilate the epistemic effect of the triggered action into the knowledge base, and if it is
   1) a physical action, execute it by calling the associated procedure.
   2) a communicative action, execute it by sending the corresponding message to the specified addressee.

The following *cycle* procedure is a Prolog-style meta-logic specification of the reagent execution model.

```
cycle( KB)
← newEvent( Evt),
   findall( ActEff, (reaction(ActEff,Evt,Cond), demo(KB,Cond)), ActEffs),
   perform( ActEffs, KB, KB′),
   cycle( KB′).

perform( [], KB, KB).

perform( [Act/Eff | ActEffs], KB, KB′)
← execute( Act),
   assimilate( Eff, KB, KB1),
   perform( ActEffs, KB1, KB′).

execute( noAct).
execute( do(Act)) ← call( Act).
execute( send(Msg,To)) ← pvm_send( To, 1, Msg).
```

Here, reaction rules are represented as triples $\langle Act/Eff, Evt, Cond\rangle$ in the table *reaction*. A null action *noAct* is used to represent epistemic actions as *noAct/Eff*. An incoming event message *Evt* is popped from the message queue, and subsequently matched with suitable reaction rules. If the precondition *Cond* of a rule matching *Evt* holds in the current knowledge state, expressed by *demo(KB, Cond)*, the epistemic effect *Eff* associated with the action *Act* is assimilated into the knowledge base, the physical or communicative action *Act* is performed by means of appropriate procedure calls,[5] and *cycle* starts over with the updated knowledge base KB′. The *demo* and *assimilate* meta-predicates are formally related to our knowledge system concepts of inference and update:

$$\begin{aligned} demo(\mathrm{KB}, Cond) &:\Longleftrightarrow \mathrm{KB} \vdash Cond \\ assimilate(Eff, \mathrm{KB}, \mathrm{KB}') &:\Longleftrightarrow \mathrm{KB}' = \mathsf{Upd}(\mathrm{KB}, Eff) \end{aligned}$$

4 Secure Inter-Agent Communication

Similar to the KQML model of communication[6], we assume that the following requirements are met by any vivid agent system:

- Agents may interact asynchronously with more than one other agent at the same time.
- Agents are known to one another by their symbolic names, rather than their IP addresses. There may be special agents, called *facilitators*, which provide address information services in order to facilitate communication.
- An agent communicates verbally with other agents: actively by sending, and passively by receiving, typed messages.[7]

[5] The above realization of communication acts is based on the built-in *pvm_send* of PVM-Prolog.

[6] See, e.g., [Lab96].

[7] In addition, there may be non-verbal forms of communication, e.g. by means of perception.

- Messages may be sent over network links, or via specific radio links, or, similar to human communication, by means of audio signals. The transport mechanism is not part of the communication model of vivid agents. Certain assumptions about message passing, however, are necessary or useful:
  - When an agent sends a message, it directs that message to a specific addressee.
  - When an agent receives a message, it knows the sender of that message.
  - The order of messages in point-to-point communication is preserved.
  - No message gets lost.
- Message types are defined by a *communication event language* based on speech act theory.
- The arguments of a message (i.e. the 'propositional content' of the corresponding communication act) may affect the mental state of both the sender and the receiver.

Communication in multiagent systems should be based on the *speech act theory* of Austin and Searle [Aus62, Sea69], an informal theory within analytical philosophy of language. The essential insight of speech act theory was that an utterance by a speaker is, in general, not the mere statement of a true or false sentence, but rather an *action* of a specific kind (such as an assertion, a request, a promise, etc.). Therefore, logic alone is not sufficient for a semantic account of verbal communication.

In our model of agents, the semantics of communicative actions is rather determined by

1. a mentalistic model of agents, defining their *mental state*, together with a notion of mental conditions and mental effects of actions,
2. a satisfaction relation between mental states and mental conditions,
3. an operation that assimilates mental effects into a mental state,
4. the assignment of a mental precondition and a mental effect to each action, and
5. associating with each type of communicative action a type of reaction (of the addressee of a communication act).

In this paper, we use the simple model of *reagents*, where the mental state consists only of beliefs (represented in a KB), the mental satisfaction relation and the mental assimilate operation are $\vdash$ and Upd, and communicative actions are represented by means of reaction rules.

We use a functional predicate

agent(Agent, Clearance),

for recording the clearance levels of all agents known to an agent. Agents without an entry in this table will be assigned clearance level 0 (unclassified) by default. The following definition of secure communication is based on the assumption that all agents involved in the communication are believed to be truthful and

competent by their fellow agents, implying that they normally provide correct information.

The basic inter-agent communication functionality consists of three types of communication events: tell, ask, and reply.

4.1 Tell

The piece of information conveyed by a *tell* act is assimilated into the beliefs of the receiver (according to the above definition of secure update).

This is expressed by the following reaction rules:

$$r_1 : F/C \leftarrow \mathsf{recvMsg}[\mathrm{tell}(F), S],\ \mathit{agent}(S, C)$$
$$r_2 : F/0 \leftarrow \mathsf{recvMsg}[\mathrm{tell}(F), S],\ \neg\exists C(\mathit{agent}(S, C))$$

where F is a formula representing an admissible input, i.e. it has either the form of an atom, $r(\mathbf{c})$, or a negated atom, $\neg r(\mathbf{c})$, possibly prefixed by a subscripted belief operator B_λ, and C is the clearance level of the sender S (F, C, and S are logical variables like in Prolog).

4.2 Ask

Since only agents with incomplete information will ask other agents, we may assume that the knowledge system of an asking agent is $\boldsymbol{F}$, the system of relational factbases, or any conservative extension of it. Agents with knowledge systems such as $\boldsymbol{A}$, or $S\boldsymbol{A}$, have complete information, i.e. for any if-query F, they believe either F or $\neg F$. Such agents will be asked for information by other agents, and they will reply to them, but they will never ask themselves.

For practical reasons, each query is associated with an ID, called *query handle*. This ID is used to store queries in the system table *query* until the answers are received, or until they are timed out. Like in KQML, we distinguish between

1. **askif**: asking an if-query,
2. **askone**: asking for one (possibly non-deterministic) answer substitution, like in Prolog, and
3. **askall**: asking for all answer substitutions (i.e. a table), like in SQL.

The reaction rules for handling an ask-if event use the meta-predicate *ifans(IfQuery, Answer)* which holds in the current knowledge state X, whenever $\mathsf{Ans}(X, \mathit{IfQuery}) = \mathit{Answer}$.

$$r_3 : \mathsf{sendMsg}[\mathrm{replyif}(\mathit{QID}, A), S] \leftarrow \mathsf{recvMsg}[\mathrm{askif}(\mathit{QID}, F), S],\ \mathit{agent}(S, C) \wedge \mathit{ifans}(F/C, A)$$

$$r_4\ \mathsf{sendMsg}[\mathrm{replyif}(\mathit{QID}, A), S] \leftarrow \mathsf{recvMsg}[\mathrm{askif}(\mathit{QID}, F), S],\ \neg\exists C(\mathit{agent}(S, C)) \wedge \mathit{ifans}(F/0, A)$$

where F is a formula representing a relational database if-query, and the answer value A is either yes or no. The reactions to ask-all and ask-one events are defined in a similar way:

$$r_5 : \mathsf{sendMsg}[\text{replyall}(QID, A), S] \\ \leftarrow \mathsf{recvMsg}[\text{askall}(QID, F), S],\ agent(S, C) \wedge allans(F/C, A)$$

$$r_6 : \mathsf{sendMsg}[\text{replyone}(QID, A), S] \\ \leftarrow \mathsf{recvMsg}[\text{askone}(QID, F), S],\ agent(S, C) \wedge oneans(F/C, A)$$

using the meta-predicates *allans* and *oneans* providing an answer set, resp. a single answer substitution. We have omitted the rules for the case of a sender without an entry in the *agent* table, since they are analogous to the corresponding rule above.

4.3 Reply

Replies to an if-query are processed as follows:

$$r_7 : F \leftarrow \mathsf{recvMsg}[\text{replyif}(QID, \mathsf{yes}), S],\ query(QID, F)$$
$$r_8 : \neg F \leftarrow \mathsf{recvMsg}[\text{replyif}(QID, \mathsf{no}), S],\ query(QID, F)$$

4.4 An Example of Secure Communication

In our final example, we assume three information agents:

1. A software agent d serving as the personal assistant to the doctor in charge of treating MJ and BY. This agent works with a relational database which does not have complete information about diagnoses. Its initial state is $X_d^0 = \{d(BY, alc)\}$.
2. A software agent s serving as the personal assistant to a secretary working in the hospital administration.
3. The hospital MLS database X_{hosp} from above, which reacts to tell and ask events by assimilating new inputs and replying to queries. Its agent name is *hdb* (hospital database). It uses the security classifications *agent(s,1)* and *agent(d,2)*.

Case 1: Assume that a reporter asks the secretary (by email) whether MJ is currently a patient in the hospital. The personal assistant of the secretary, although knowing that $p(MJ)$, asks *hdb* whether the reporter may know that fact by sending the message $\text{askif}(\mathsf{B}_0 p(MJ))$, firing at *hdb* the reaction rule r_3 with the following instantiation (we omit the query ID):

$$\mathsf{sendMsg}[\text{replyif}(\mathsf{no}), s] \\ \leftarrow \mathsf{recvMsg}[\text{askif}(\mathsf{B}_0 p(MJ)), s],\ agent(s, 1) \wedge ifans((\mathsf{B}_0 p(MJ))/1, \mathsf{no})$$

since $\mathsf{Ans}(X^0_{hosp}, (\mathsf{B}_0 p(MJ))/1) = \mathsf{no}$. After receiving this negative answer, s forwards it to the reporter.

Case 2: Assume that the doctor is wondering if BY was diagnosed to have hepatitis by one of her colleagues, and thus sends the message $\mathit{askif}(Q2, d(BY, hep)$ to hdb, recording the query together with its ID $Q2$ as the fact $\mathit{query}(Q2, d(BY, hep))$. This triggers at hdb the rule

$$\mathsf{sendMsg}[\mathrm{replyif}(Q2, \mathsf{no}), d] \\ \leftarrow \mathsf{recvMsg}[\mathrm{askif}(Q2, d(BY, hep)), d],\ \mathit{agent}(d, 2) \wedge \mathit{ifans}(d(BY, hep))/2, \mathsf{no})$$

since $\mathsf{Ans}(X^0_{hosp},\ d(BY, hep)/2) = \mathsf{no}$. Agent d reacts to the reply by applying the rule r_7:

$$\neg d(BY, hep) \leftarrow \mathsf{recvMsg}[\mathrm{replyif}(Q2, \mathsf{no}), \mathit{hdb}],\ \mathit{query}(Q2, d(BY, hep))$$

yielding the update

$$X^1_d = \mathsf{Upd}(X^0_d, \neg d(BY, hep)) = \{d(BY, alc)\}$$

Notice that X_d, as a relational database, is not capable of recording the negative information $\neg d(BY, hep)$, although it would be useful for avoiding further questions (communication overload) in the style of replication. This indicates that we have to extend the concept of relational databases in the frmework of multidatabase systems.

5 Conclusion

We have shown how the concept of multi-level security can be applied in vivid agent systems. For this purpose, we have defined

1. the knowledge system of MLS databases including a formalization of the "no read/write up" security model, and
2. basic inter-agent communication rules which take security classifications into consideration.

A more realistic treatment would have to account for MLS databases with integrity constraints (such as functional dependencies), where 'cover stories' are overridden by attribute values with a higher classification because of the inconsistency created by the violation of functional dependencies. This can be achieved by a straightforward extension of our present model.

Acknowledgment The author is grateful to the referees whose comments and suggestions have helped to improve the paper.

References

[Aus62] J.L Austin. *How to Do Things with Words.* Harvard University Press, Cambridge (MA), 1962.

[JS91] S. Jajodia and R. Sandhu. Toward a multilevel secure relational data model. In *Proc. ACM SIGMOD*, pages 50–59. ACM Press, 1991.

[Lab96] Y. Labrou. *Semantics for an Agent Communication Language.* PhD thesis, University of Maryland Graduate School, 1996.

[Lan81] C.E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, 1981.

[Sea69] J.R. Searle. *Speech Acts.* Cambridge University Press, Cambridge (UK), 1969.

[SWQ94] K. Smith, M. Winslett, and X. Qian. Formal query languages for secure relational databases. *ACM Transactions on Database Systems*, 19(4):626–662, 1994.

[Wag95] G. Wagner. From information systems to knowledge systems. In W. Hesse E.D. Falkenberg and A. Olivé, editors, *Information System Concepts*, pages 179–194, London, 1995. Chapman & Hall.

[Wag96] G. Wagner. A logical and operational model of scalable knowledge- and perception-based agents. In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away (Proc. of MAAMAW'96)*, pages 26–41. Springer-Verlag, LNAI 1038, 1996.

[Wag97] G. Wagner. Conceptual foundations of artificial agents. Habilitation thesis, Univ. Leipzig, 1997.

This article was processed using the LaTeX macro package with LLNCS style